TP4 – Classification bayésienne

Réalisé par : Zahra BENSLIMANE

Chargé de TP : Flavien Lebrun

Chargée de cours : Catherine Achard

Dans ce TP, nous souhaitons résoudre un problème de classification binaire en utilisant une méthode de classification bayésienne dans laquelle, chaque classe va être modélisée en utilisant la méthode « Kernel density Estimation »

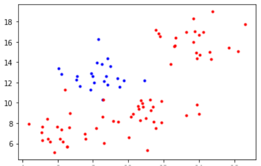
I. Chargement et visualisation des données

Charger les données (TP4.npy) et visualiser les. Combien y a-t-il de points dans la base d'apprentissage ? Dans la base de test ? Quelle est la dimension des données ?

```
# Chargement des librairies
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import math


[X_train, y_train, X_test, y_test] = np.load('TP4.npy',allow_pickle=True)
print("X_train.shape = ", X_train.shape )
print("y_train.shape = ",y_train.shape)
print("X_test.shape = ", X_test.shape)
print("y_test.shape = ",y_test.shape)
print("La dimension des datas [Train+Test]: 120")
print("******************************************\n")

# Classificatio binaire : same as TP3
T_train = X_train[np.where(y_train==1),:]
T_train = np.reshape(T_train,(T_train.shape[1],T_train.shape[2] ))
#Pixel non peau
F_train = X_train[np.where(y_train==0),:]
F_train = np.reshape(F_train,(F_train.shape[1],F_train.shape[2] ))

print("T_train.shape = ",T_train.shape, "#Pixel True : peau : Deux composantes Cb et Cr")
print("F_train.shape = ",F_train.shape , "#Pixel  False : Non peau  : Deux composantes Cb
print("******************************************\n")

plt.plot(F_train[:,0], F_train[:,1], '.b')
plt.show
plt.plot(T_train[:,0], T_train[:,1], '.r')
plt.show
```



```
X_train.shape =  (90, 2)
y_train.shape =  (90,)
X_test.shape =  (30, 2)
y_test.shape =  (30,)
La dimension des datas [Train+Test]: 120
******************************************

T_train.shape =  (68, 2) #Pixel True : peau : Deux composantes Cb et Cr
F_train.shape =  (22, 2) #Pixel  False : Non peau  : Deux composantes Cb et Cr
******************************************

<function matplotlib.pyplot.show>
```



II. Estimation des densités de probabilité:

Ecrire une fonction ddp_parzen($\boldsymbol{x}$,$\boldsymbol{X}$, $\boldsymbol{Cov}$) qui estime la densité de probabilité en un point $\boldsymbol{x}$ à partir des points de la base d'apprentissage $\boldsymbol{X}$, en utilisant un noyau 2D gaussien. On utilisera une matrice de covariance diagonale avec un écart type identique pour les deux dimensions. On pourra utiliser la fonction norm2() développée au TP3.

```
mCb = np.mean(T_train[:,0])    # moyennes et écarts-type de la composante Cb
mCr = np.mean(T_train[:,1])    # moyennes et écarts-type de la composante Cr

σCb = np.std(T_train[:,0])     # écarts-type de la composante Cb
σCr = np.std(T_train[:,1])     # écarts-type de la composante Cr
print("****************************")
print("Les moyennes :\nmCb = ", mCb, "\nmCr = ",mCr,"\nLes écarts-types :\nσCb = ",σCb,"
print("****************************")

# Calcul de la matrice de covariance des points chaire peau
Sigma = np.cov(np.transpose(F_train))
print("La matrice de Covariance: \n",Sigma)
```

```
****************************
Les moyennes :
mCb =  10.410166628218883
mCr =  10.659179155067964
Les écarts-types :
σCb =  3.2571959424455033
σCr =  4.042314473899821
```



```
****************************
La matrice de Covariance:
 [[ 1.29021046 -0.12672624]
 [-0.12672624  1.5211322 ]]
```

Questions: Rappeler le principe de fonctionnement de l'estimation de densité de probabilité par noyau. Que représente le noyau ?

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable.

Let (x1, x2, ..., xn) be independent and identically distributed samples drawn from some univariate distribution with an unknown density ƒ at any given point x. We are interested in estimating the shape of this function ƒ. Its kernel density estimator is

where K is the kernel — a non-negative function — and h > 0 is a smoothing parameter called the bandwidth. A kernel with subscript h is called the scaled kernel and defined as Kh(x) = 1/h K(x/h). Intuitively one wants to choose h as small as the data will allow; however, there is always a trade-off between the bias of the estimator and its variance. The choice of bandwidth is discussed in more detail below.

None parametric way : cela signifie que nous ne supposons aucune distribition sous-jacente pour nos données, par exemple, nous ne supposons pas que nos données viennent d'une gaussienne, et donc nous n'avons pas de moyenne et d'écarts types à ajuster.

Définition [modifier | modifier le code]

Si $x_1, x_2, ..., x_N \sim f$ est un échantillon i.i.d. d'une variable aléatoire, alors l'estimateur non-paramétrique par la méthode du noyau de la densité est :

$$\widehat{f}_h(x) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right)$$

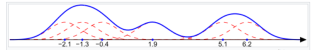
où K est un noyau (*kernel* en anglais) et h un paramètre nommé ***fenêtre***, qui régit le degré de lissage de l'estimation. Bien souvent, K est choisi comme la densité d'une fonction gaussienne standard (espérance nulle et variance unitaire) :

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Intuition [modifier | modifier le code]

La méthode de Parzen est une généralisation de la méthode d'estimation par histogramme. Dans un histogramme, la densité en un point x est estimée par la proportion d'observations $x_1, x_2, ..., x_N$ qui se trouvent à proximité de x. Pour cela, on trace une boîte en x et dont la largeur est gouvernée par un paramètre de lissage h ; on compte ensuite le nombre d'observations qui appartiennent à cette boîte. Cette estimation, qui dépend du paramètre de lissage h, présente de bonnes propriétés statistiques mais est par construction non-continue.

La méthode du noyau consiste à retrouver la continuité : pour cela, on remplace la boîte centrée en x et de largeur h par une gaussienne centrée en x. Plus une observation est proche du point de support x plus la courbe en cloche lui donnera une valeur numérique importante. À l'inverse, les observations trop éloignées de x se voient affecter une valeur numérique négligeable. L'estimateur est formé par la somme (ou plutôt la moyenne) des courbes en cloche. Comme indiqué sur l'image suivante, il est clairement continu.



Six courbes en cloche gaussiennes (rouge) et leur somme (bleu). L'estimateur à noyau de la densité $f(x)$ est en fait la moyenne (on divise par le nombre de courbes en cloche, 6). La variance des normales est posée à 0,5. Notons enfin que plus il y a d'observations dans le voisinage d'un point, plus sa densité est élevée.

```
# Fonction qui renvoie la probabilité en un point x
```

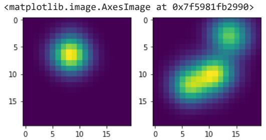


```
def norm2(x, m, cov):
  a = np.dot(np.transpose((x-m)), np.linalg.inv(cov))
  a = np.dot(a, (x-m))
  p =1/(math.sqrt(2*math.pi*np.linalg.det(cov)))*math.exp(-0.5*a)
  return p
# estime la densité de probabilité en un point x à partir des points de la base d'apprenti
def ddp_parzen(x,X, Cov):
  somme = sum(norm2(x,Xi, Cov) for Xi in X )
  return  somme/X.shape[0]
s = 2
# Choix de la matrice de covariance
Cov = np.array([[s*s, 0],[0,s*s]])
print("cov = \n", Cov)
```

```
cov =
 [[4 0]
 [0 4]]
```

```
p0 = np.zeros((20,20))
p1 = np.zeros((20,20))
for i in range(20):
  for j in range(20):
        p0[j,i] = ddp_parzen(np.transpose([i,j]),F_train, Cov)
        p1[j,i] = ddp_parzen(np.transpose([i,j]),T_train, Cov)
plt.figure()
plt.subplot(1,2,1); plt.imshow(np.flipud(p0))
plt.subplot(1,2,2); plt.imshow(np.flipud(p1))
```

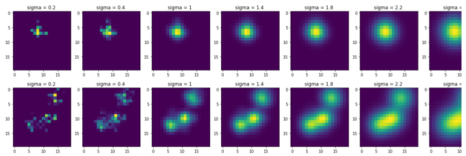
```
<matplotlib.image.AxesImage at 0x7f5981fb2990>
```



```
sigma = [0.2,0.4,1,1.4,1.8,2.2,2.8,3.2]
fig = plt.figure(figsize = (25,7))
for index,s in enumerate(sigma):
  for i in range(20):
    for j in range(20):
        p0[j,i] = ddp_parzen(np.transpose([i,j]),F_train, np.array([[s*s, 0],[0,s*s]]))
        p1[j,i] = ddp_parzen(np.transpose([i,j]),T_train, np.array([[s*s, 0],[0,s*s]]))
  plt.subplot(2,len(sigma),index+1) ; plt.imshow(np.flipud(p0)); plt.title("sigma = {} ".f

fig = plt.figure(figsize = (25,7))
for index,s in enumerate(sigma):
  for i in range(20):
    for j in range(20):
        p0[j,i] = ddp_parzen(np.transpose([i,j]),F_train, np.array([[s*s, 0],[0,s*s]]))
```



```
        p1[j,i] = ddp_parzen(np.transpose([i,j]),T_train, np.array([[s*s, 0],[0,s*s]]))
  plt.subplot(2,len(sigma),index+1); plt.imshow(np.flipud(p1)) ; plt.title("sigma = {} ".f
```



A priori et sans faire le test, LA valeur de sigma que je pense adéquate pour faire la classification est sigma = 1

III. Classification bayésienne

DDP à priori de chaque classe binaire

```
print("DDP à priori de C1 : P1 = ",T_train.shape[0]/X_train.shape[0])  #Le nombre de pixel
print("DDP à priori de C0 : P0 = ",F_train.shape[0]/X_train.shape[0])  #Le nombre de pixel
```

```
DDP à priori de C0 : P0 =  0.7555555555555555
DDP à priori de C1 : P1 =  0.24444444444444444
```

Réaliser la classification des points $\boldsymbol{x}$ de la base de test en utilisant la règle de bayes

$C = argmax_i[p(Ci/\boldsymbol{x})] = argmax_i[ddp\_parzen(\boldsymbol{x}, \boldsymbol{X}i, \boldsymbol{Cov} * Pi)]$

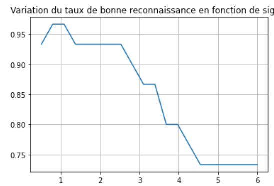
Où $\boldsymbol{X}i$ est l'ensemble des exemples d'apprentissage de la classe Ci

Calcul de la DDP des exemples de test

```
# Densités à priori des classses
p_C1 = T_train.shape[0]/X_train.shape[0]
p_C0 = F_train.shape[0]/X_train.shape[0]

sigmaTab = np.linspace(start=0.5,stop=6,num=20)
#sigmaTab = [1] # test
```



```
# Calcul des DDP des x exemples de test
p0_test, p1_test = [],[]
# Taub de bonne reconnaissance
accuracy = np.zeros(len(sigmaTab))
# Stocker les matrices de confusions

for indx,sigma in enumerate(sigmaTab):
    cov = (sigma**2)*np.eye(2)

    y_predict=np.zeros(X_test.shape[0])
    for i,xi in enumerate(X_test):
        p0_test = ddp_parzen(xi,F_train,cov)  #vraissemblance p(x|y= C0)
        p1_test = ddp_parzen(xi,T_train, cov) #vraissemblance p(x|y= C1)

        # Probabilités à postériori
        p0_aposteriori = p0_test * p_C0   # p(y=C0 |x)
        p1_aposteriori = p1_test * p_C1   # p(y=C1 |x)
        # Classification :  Régle du maximum à posteriori

        if p1_aposteriori > p0_aposteriori:
          y_predict[i] = 1
        else:
          y_predict[i] = 0  # Or do nothing

    conf_matrix = confusion_matrix(y_test,y_predict)
    # Taut de bonne reconnaissance
    accuracy[indx]=np.sum(np.diag(conf_matrix))/np.sum(conf_matrix)
    #print(sigma, accuracy[indx])

plt.plot(sigmaTab,accuracy)
plt.title("Variation du taux de bonne reconnaissance en fonction de sigma")
plt.grid()
plt.show()
```



Afficher le taux de reconnaissance. Faire varier et optimiser le taux de reconnaissance. Questions Pour quelle valeur de obtient-on la meilleure classification ? Etait-ce prévisible ? Expliquer le résultat.